ECE521:

Week 11, Lecture 20

27 March 2017:
HMM learning/inference

With thanks to Russ Salakhutdinov

- **Examples of other perspectives**

  - Murphy 17.4
  - End of Russell & Norvig 15.2
    (*Artificial Intelligence: A Modern Approach*)
  - Bishop 13.2.5, 8.4.4

Outline

- **HMM learning and inference:**
  1) Probability of an observed data sequence
  2) Learning the model parameters
  3) Inferring the most likely state sequence

- **Thursday:** Message passing
  - Introduction
  - Sum-product algorithm

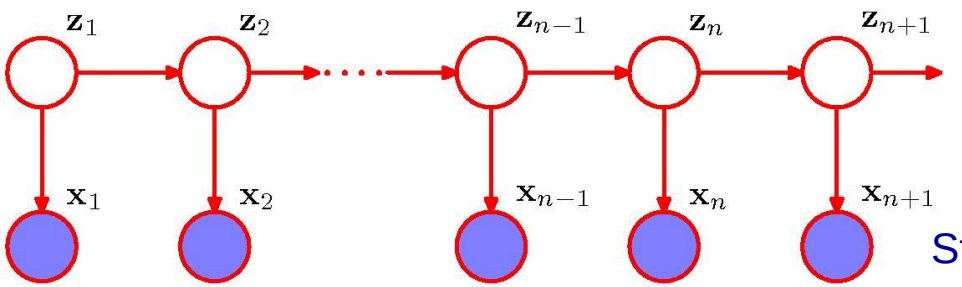
Recap of Hidden Markov Models

- First-order Markov chain generates hidden state sequence (known as transition probabilities):

$$p(\mathbf{z}_n = k | \mathbf{z}_{n-1} = j) = A_{jk}, \qquad p(\mathbf{z}_1 = k) = \pi_k.$$

- A set of output probability distributions (**one per state**) converts state path into sequence of observable symbols/vectors (known as emission probabilities): $p(\mathbf{x}_n | \mathbf{z}_n, \boldsymbol{\phi})$.

Can be e.g. Gaussian if **x** is continuous.
Conditional probability table if **x** is discrete.



State transition Observation model

$$p(\mathbf{x}_1, ..., \mathbf{x}_N, \mathbf{z}_1, .., \mathbf{z}_N) = p(\mathbf{z}_1) \prod_{n=2}^{N} p(\mathbf{z}_n | \mathbf{z}_{n-1}) \prod_{n=1}^{N} p(\mathbf{x}_n | \mathbf{z}_n).$$

Recap of Hidden Markov Models

In the last lecture you looked at six examples:

- Sampling (generating states & observations)
- Prediction of the next state(s) given the current state (Example 2)
- Inferring the latent states behind an observed sequence (Example 6)

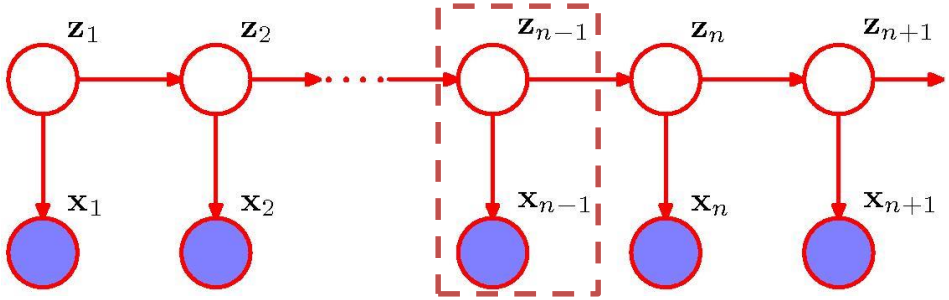
We build on Example 6 today

Three problems

- The joint distribution over the observed- and latent variables is given by:

$$p(\mathbf{X}, \mathbf{Z}|\theta) = p(\mathbf{z}_1|\boldsymbol{\pi}) \prod_{n=2}^{N} p(\mathbf{z}_n|\mathbf{z}_{n-1}, A) \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\phi}),$$

where $\theta = \{\boldsymbol{\pi}, A, \boldsymbol{\phi}\}$ are the model parameters.

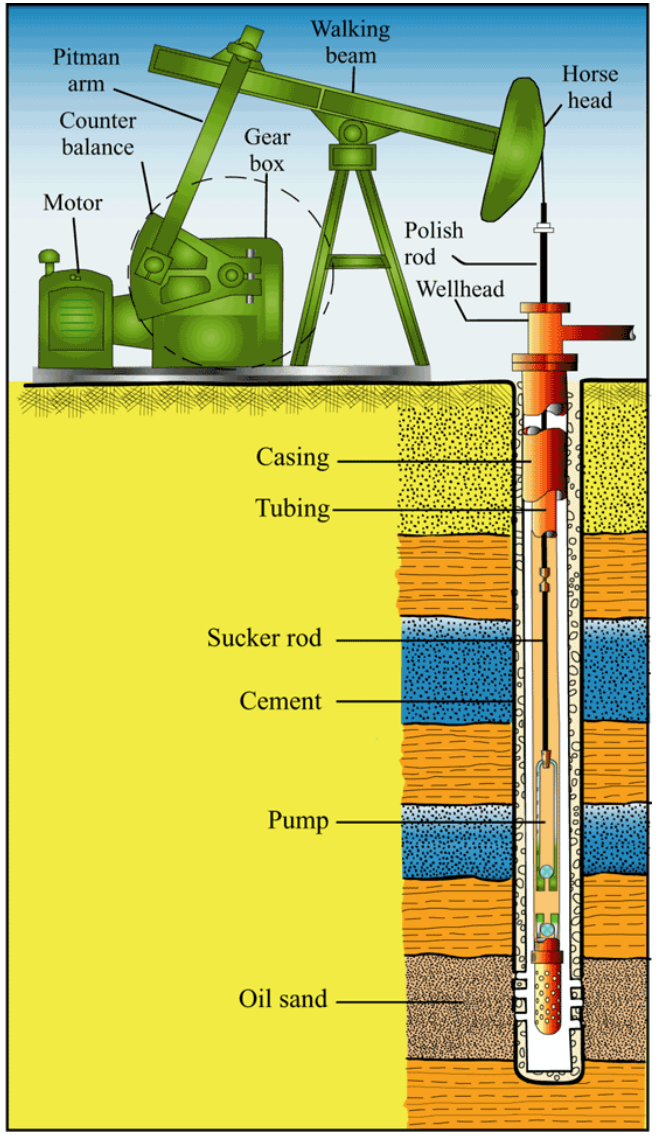


Three problems and three solutions:

1. Computing probabilities of observed sequences: *Forward-backward algorithm*
2. Learning of parameters: *Baum-Welch algorithm*
3. Inference of hidden state sequences: *Viterbi algorithm* ←

Viterbi algorithm application: *Casing Running*

www.youtube.com/watch?v=F-HrLO5m_-s



Outline

- HMM learning and inference:
  1) **Probability of an observed data sequence**
  2) Learning the model parameters
  3) Inferring the most likely state sequence

Three problems and three solutions:

1. Computing probabilities of observed sequences: *Forward-backward algorithm*
2. Learning of parameters: *Baum-Welch algorithm*
3. Inference of hidden state sequences: *Viterbi algorithm*

Maximum Likelihood for the HMM

- We observe a dataset $\mathbf{X} = \{\mathbf{x}_1,\ldots,\mathbf{x}_N\}$.
- The goal is to determine model parameters $\theta = \{\boldsymbol{\pi}, A, \boldsymbol{\phi}\}$.
- The probability of an observed sequence takes the form:

$$p(\mathbf{X}|\theta) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\theta).$$

$$p(\text{observed sequence}) = \sum_{\text{all paths}} p(\text{observed outputs, state paths}).$$

- Recall that, in contrast to mixture models, the joint distribution $p(\mathbf{X},\mathbf{Z} \mid \mu)$ does not factorize over n.
- It looks hard: N variables, each of which has K states. Hence K^N total paths.

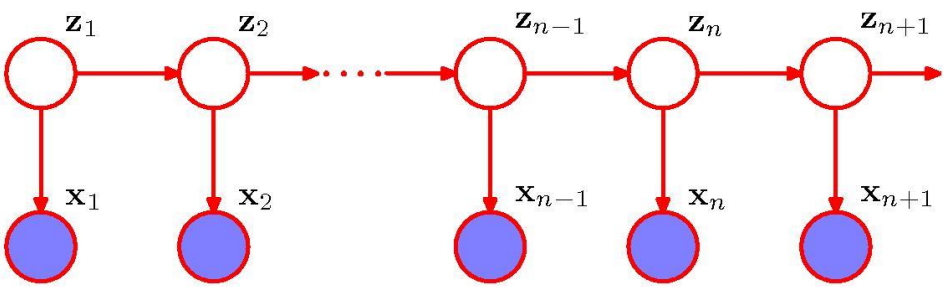
Probability of an Observed Sequence

- Recalling slides 22-23 from Thursday (lecture 19), probabilities factorize:

$$p(\mathbf{X}|\theta) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\theta) = \sum_{\mathbf{z}_1,\ldots,\mathbf{z}_n} p(\mathbf{z}_1, \mathbf{x}_1) \prod_{n=2}^{N} p(\mathbf{z}_n|\mathbf{z}_{n-1}) p(\mathbf{x}_n|\mathbf{z}_n)$$

$$= \sum_{\mathbf{z}_1} p(\mathbf{z}_1) p(\mathbf{x}_1|\mathbf{z}_1) \sum_{\mathbf{z}_2} p(\mathbf{z}_2|\mathbf{z}_1) p(\mathbf{x}_2|\mathbf{z}_2) \ldots$$

$$\sum_{\mathbf{z}_N} p(\mathbf{z}_N|\mathbf{z}_{N-1}) p(\mathbf{x}_N|\mathbf{z}_N).$$



- Dynamic Programming: By moving the summations inside, we can save a lot of work.

EM algorithm for HMMs: overview

- We cannot perform direct maximization (no closed-form solution):

$$p(\mathbf{X}|\theta) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\theta).$$

- EM algorithm: we will derive an efficient algorithm for maximizing the likelihood function in HMMs (and, later, in linear state-space models).

- E-step: Compute the posterior distribution over latent variables:

$$p(\mathbf{Z}|\mathbf{X}, \theta^{old}).$$

- M-step: Maximize the expected complete data log-likelihood:

$$Q(\theta, \theta^{old}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta^{old}) \log p(\mathbf{X}, \mathbf{Z}|\theta).$$

- If we knew the true state path, then ML parameter estimation would be trivial.

- We will first look at the E-step: Computing the true posterior distribution over the state paths.

Inference of Hidden States

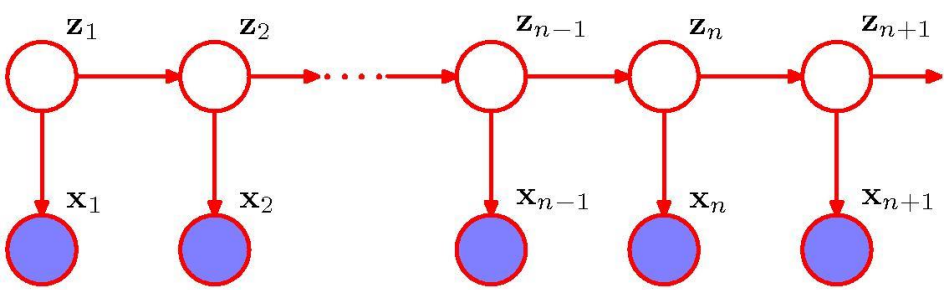
- We want to estimate the hidden states given observations. To start with, let us estimate a single hidden state:

$$\gamma(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{z}_n)p(\mathbf{z}_n)}{p(\mathbf{X})}.$$

- Using the conditional-independence property, we obtain:

$$p(\mathbf{z}_n|\mathbf{X}) = \frac{p(\mathbf{x}_1, .., \mathbf{x}_n|\mathbf{z}_n)p(\mathbf{x}_{n+1}, ..., \mathbf{x}_N|\mathbf{z}_n)p(\mathbf{z}_n)}{p(\mathbf{X})}$$

$$= \frac{p(\mathbf{x}_1, .., \mathbf{x}_n, \mathbf{z}_n)p(\mathbf{x}_{n+1}, ..., \mathbf{x}_N|\mathbf{z}_n)}{p(\mathbf{X})} = \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})}.$$



Inference of Hidden States

- Hence:

$$\gamma(\mathbf{z}_n) = \frac{p(\mathbf{x}_1, .., \mathbf{x}_n, \mathbf{z}_n)p(\mathbf{x}_{n+1}, ..., \mathbf{x}_N|\mathbf{z}_n)}{p(\mathbf{X})} = \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})}.$$

$$\alpha(\mathbf{z}_n) \equiv p(\mathbf{x}_1, .., \mathbf{x}_n, \mathbf{z}_n)$$

The joint probability of observing all of the data up to time *n*, and $\mathbf{z}_n$.

$$\beta(\mathbf{z}_n) \equiv p(\mathbf{x}_{n+1}, .., \mathbf{x}_N|\mathbf{z}_n).$$

The conditional probability of all future data from time *n*+1 to *N*.

- Each $\alpha(\mathbf{z}_n)$ and $\beta(\mathbf{z}_n)$ represent a set of *K* numbers, one for each of the possible settings of the 1-of-*K* binary vector $\mathbf{z}_n$.

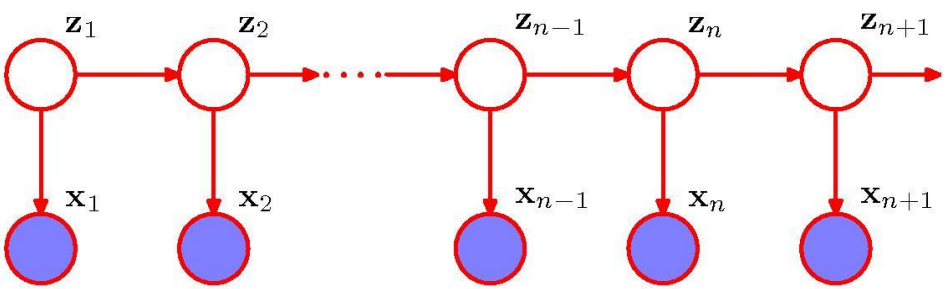
- We will derive an efficient recursive algorithm, known as the alpha-beta recursion, or forward-backward algorithm.

The Forward (α) Recursion

- The forward recursion:

$$
\begin{aligned}
\alpha(\mathbf{z}_n) &= p(\mathbf{x}_1, .., \mathbf{x}_n, \mathbf{z}_n) \\
&= p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{x}_1, .., \mathbf{x}_{n-1}, \mathbf{z}_n) \\
&= p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{x}_1, .., \mathbf{x}_{n-1}, \mathbf{z}_{n-1}, \mathbf{z}_n) \\
&= p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{x}_1, .., \mathbf{x}_{n-1}, \mathbf{z}_{n-1})p(\mathbf{z}_n|\mathbf{z}_{n-1}) \\
&= p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1})p(\mathbf{z}_n|\mathbf{z}_{n-1})
\end{aligned}
$$

Computational cost scales as $O(K^2)$.



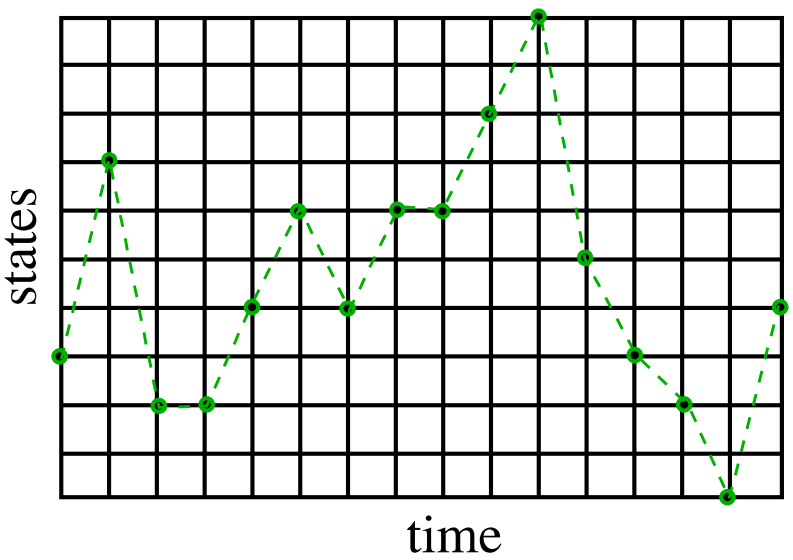
- Note that:

$$p(\mathbf{X}) = \sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N).$$

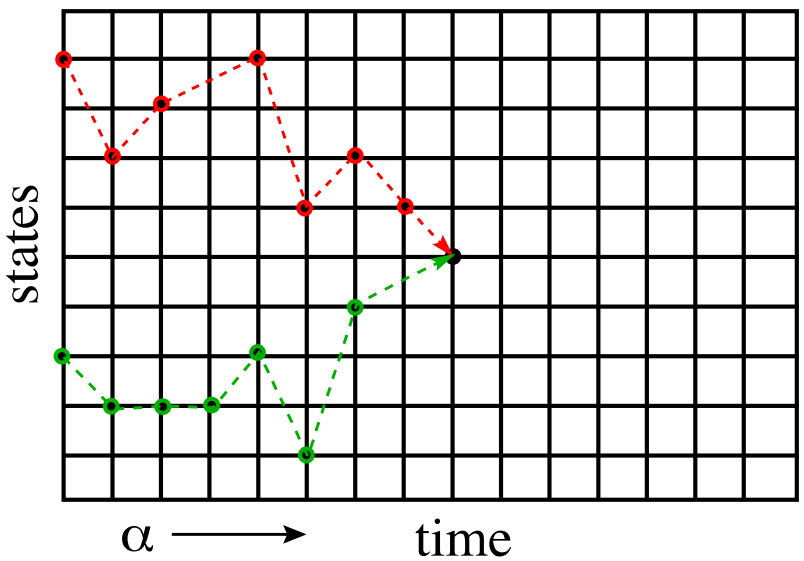
- This enables us to easily (cheaply) compute the desired likelihood.

The Forward (α) Recursion

- The forward recursion:



Exponentially many paths.

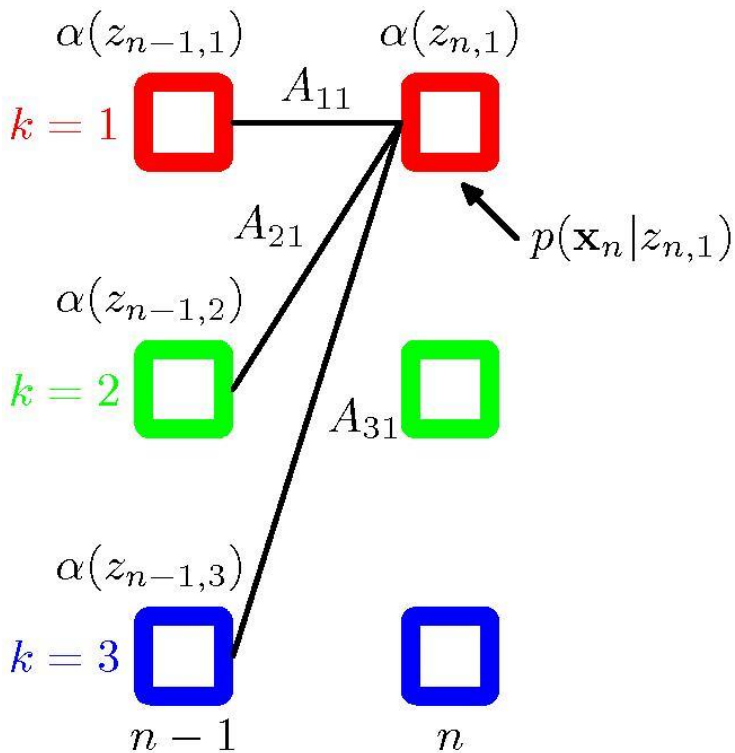


At each node, sum up the values of all incoming paths.

- This is exactly dynamic programming.

The Forward (α) Recursion

- Illustration of the forward recursion



Here $\alpha(z_{n,1})$ is obtained by:

- Taking the elements $\alpha(z_{n-1},j)$
- Summing them up with weights A_{j1}, corresponding to $p(\mathbf{z}_n \mid \mathbf{z}_{n-1})$
- Multiplying by the data contribution $p(\mathbf{x}_n \mid z_{n,1})$.

$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

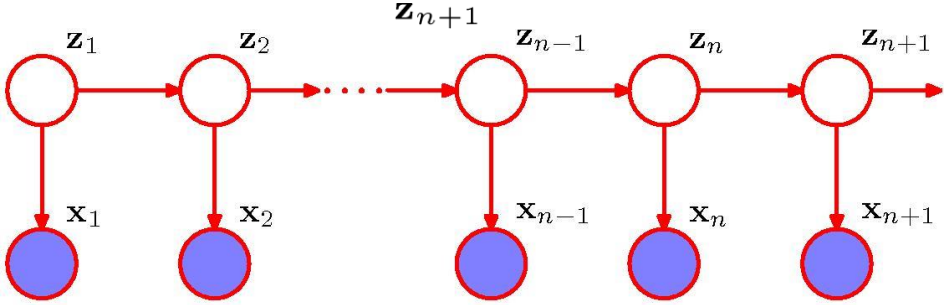
- The initial condition is given by:

$$\alpha(\mathbf{z}_1) = p(\mathbf{x}_1|\mathbf{z}_1)p(\mathbf{z}_1) = \prod_{k=1}^{K} \left[\pi_k p(\mathbf{x}_1|\boldsymbol{\phi}_k)\right]^{z_{1k}}.$$

The Backward (β) Recursion

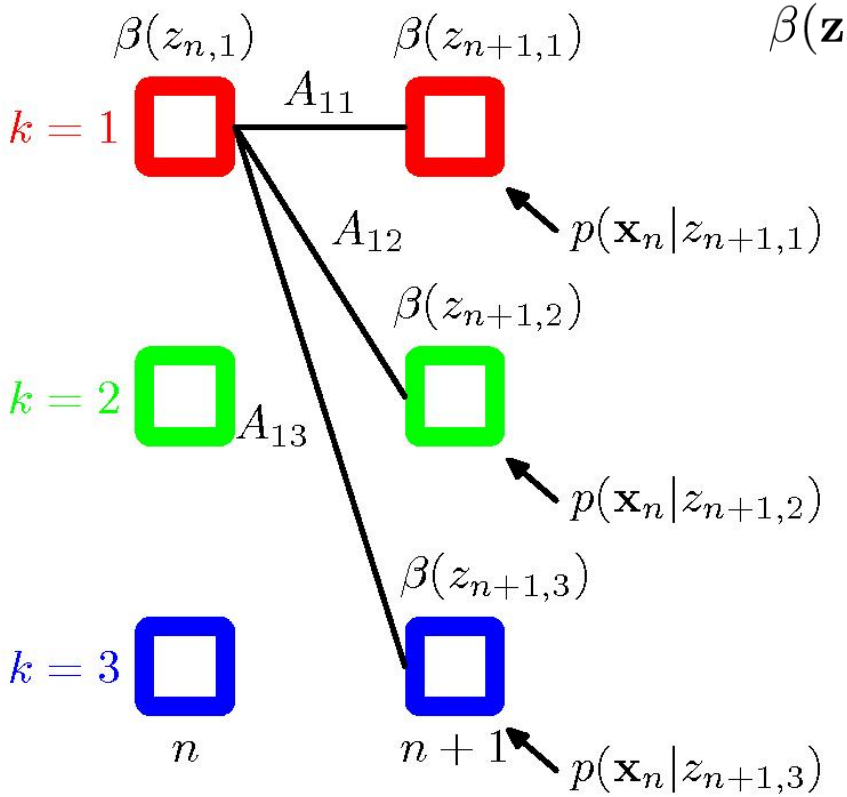
- There is also a simple recursion for $\beta(\mathbf{z}_n)$:

$$
\begin{aligned}
\beta(\mathbf{z}_n) &= p(\mathbf{x}_{n+1}, .., \mathbf{x}_N | \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, ..., \mathbf{x}_N, \mathbf{z}_{n+1} | \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, ..., \mathbf{x}_N | \mathbf{z}_{n+1}, \mathbf{z}_n) p(\mathbf{z}_{n+1} | \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, ..., \mathbf{x}_N | \mathbf{z}_{n+1}) p(\mathbf{z}_{n+1} | \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+2}, ..., \mathbf{x}_N | \mathbf{z}_{n+1}) p(\mathbf{x}_{n+1} | \mathbf{z}_{n+1}) p(\mathbf{z}_{n+1} | \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1} | \mathbf{z}_{n+1}) p(\mathbf{z}_{n+1} | \mathbf{z}_n)
\end{aligned}
$$



The Backward (β) Recursion

- Illustration of the backward recursion



$$\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n)$$

- Initial condition:

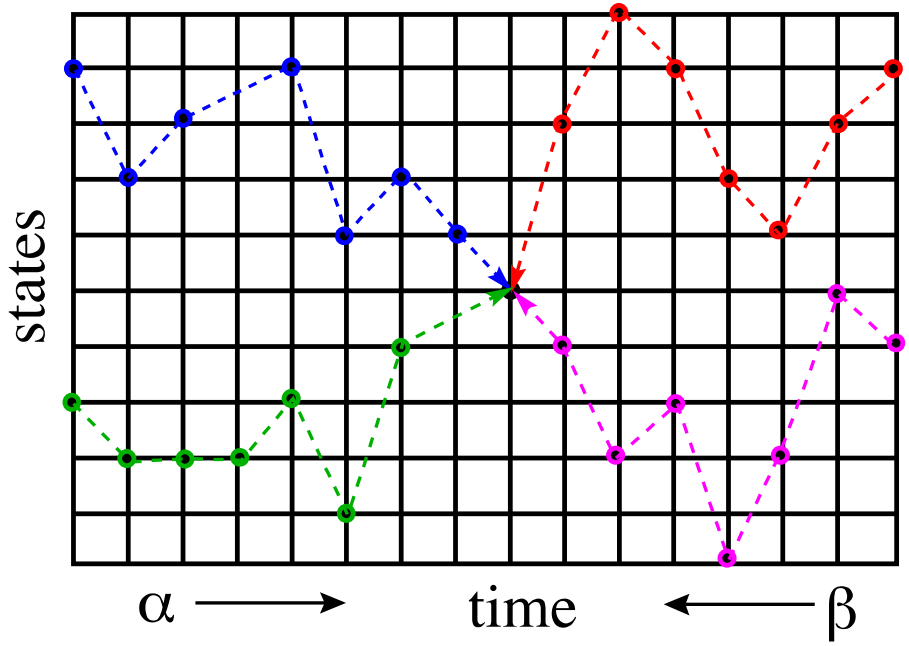
$$\begin{aligned} p(\mathbf{z}_N|\mathbf{X}) &= \frac{\alpha(\mathbf{z}_N)\beta(\mathbf{z}_N)}{p(\mathbf{X})} \\ &= \frac{p(\mathbf{X},\mathbf{z}_N)\beta(\mathbf{z}_N)}{p(\mathbf{X})}. \end{aligned}$$

- Hence:

$$\beta(\mathbf{z}_N) = 1.$$

The Backward (β) Recursion

- $\alpha(z_{nk})$ gives total inflow of probability to node (n,k).
- $\beta(z_{nk})$ gives total outflow of probability.



- In fact, we can do one forward pass to compute all the $\alpha(\mathbf{z}_n)$ and one backward pass to compute all the $\beta(\mathbf{z}_n)$ and then compute any $\gamma(\mathbf{z}_n)$ we want. Total cost is $\mathcal{O}(K^2N)$.

Computing Likelihood

- Note that

$$\sum_{\mathbf{z}_n} \gamma(\mathbf{z}_n) = \sum_{\mathbf{z}_n} p(\mathbf{z}_n|\mathbf{X}) = 1.$$

- We can compute the likelihood at any time using *α*-*β* recursion:

$$p(\mathbf{X}|\theta) = \sum_{\mathbf{z}_n} \alpha(\mathbf{z}_n)\beta(\mathbf{z}_n).$$

- In the forward calculation we proposed originally, we did this at the final time step $n = N$.

$$p(\mathbf{X}|\theta) = \sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N).$$

  because $\beta(\mathbf{z}_n)=1$.

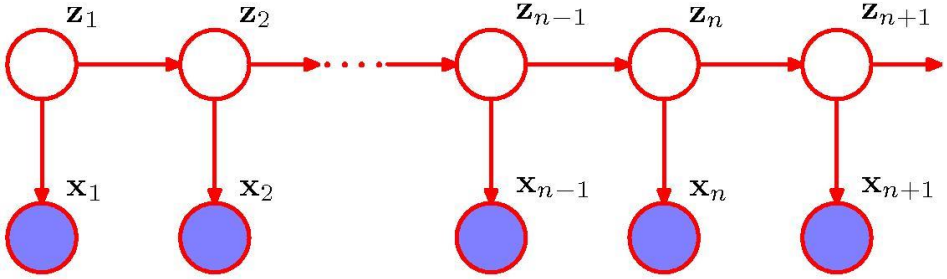
- Monitoring this quantity is a good way to check for convergence during EM

Two-Frame Inference

- We will also need the cross-time statistics for adjacent time steps:

$$
\begin{aligned}
\xi(\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{z}_{n-1}, \mathbf{z}_n | \mathbf{X}) \\
&= \frac{p(\mathbf{X}|\mathbf{z}_{n-1}, \mathbf{z}_n) p(\mathbf{z}_{n-1}, \mathbf{z}_n)}{p(\mathbf{X})} \\
&= \frac{p(\mathbf{x}_1, .., \mathbf{x}_{n-1}|\mathbf{z}_{n-1}) p(\mathbf{x}_n|\mathbf{z}_n) p(\mathbf{x}_{n+1}, ..., \mathbf{x}_N|\mathbf{z}_n) p(\mathbf{z}_n|\mathbf{z}_{n-1}) p(\mathbf{z}_{n-1})}{p(\mathbf{X})} \\
&= \frac{\alpha(\mathbf{z}_{n-1}) p(\mathbf{x}_n|\mathbf{z}_n) p(\mathbf{z}_n|\mathbf{z}_{n-1}) \beta(\mathbf{z}_n)}{p(\mathbf{X})}.
\end{aligned}
$$

- This is a *K* x *K* matrix with elements (*i*,*j*) representing the expected number of transitions from state *i* to state *j* that begin at time *n*-1, given all the observations.
- Whereas γ is the marginal posterior distribution of a latent variable, ξ is the joint posterior distribution of two successive latent variables



- It can be computed with the same α- and β recursions.

Outline

- HMM learning and inference:
  1) Probability of an observed data sequence
  2) **Learning the model parameters**
  3) Inferring the most likely state sequence

Three problems and three solutions:

1. Computing probabilities of observed sequences: *Forward-backward algorithm*
2. Learning of parameters: *Baum-Welch algorithm* ←
3. Inference of hidden state sequences: *Viterbi algorith*

The second problem: EM algorithm

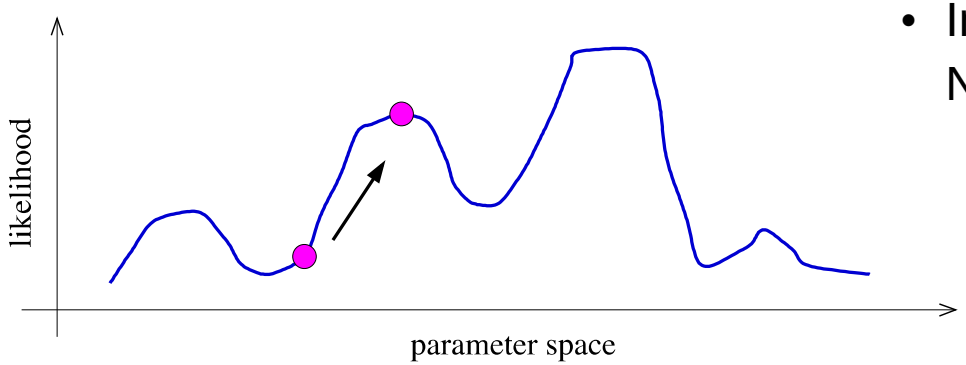
- Intuition: if only we knew the true state path then ML parameter estimation would be trivial.
- E-step: Compute the posterior distribution over the state path using *α-β* recursion (dynamic programming):

$$p(\mathbf{Z}|\mathbf{X}, \theta^{old}).$$

- M-step: Maximize the expected complete data log-likelihood (parameter re-estimation):

$$Q(\theta, \theta^{old}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta^{old}) \log p(\mathbf{X}, \mathbf{Z}|\theta).$$

- We then iterate. This is also known as a Baum-Welch algorithm (that is, EM applied to HMMs).



- In general, finding the ML parameters is NP hard, so initial conditions matter a lot

Complete Data Log-likelihood

- The complete data log-likelihood takes the form:

$$\log p(\mathbf{X}, \mathbf{Z}|\theta) = \log \left[p(\mathbf{z}_1|\boldsymbol{\pi}) \prod_{n=2}^{N} p(\mathbf{z}_n|\mathbf{z}_{n-1}, A) \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\phi}) \right]$$

$$= \log \left[\prod_{k=1}^{K} \pi_k^{z_{1k}} \prod_{n=2}^{N} \prod_{k=1}^{K} \prod_{j=1}^{K} A_{jk}^{z_{n-1,j} z_{nk}} \prod_{n=1}^{N} \prod_{k=1}^{K} p(\mathbf{x}_n|\mathbf{z}_n)^{z_{nk}} \right]$$

$$= \sum_{k=1}^{K} z_{1k} \log \pi_k + \sum_{n=2}^{N} \sum_{k=1}^{K} \sum_{j=1}^{K} [z_{n-1,j} z_{nk}] \log A_{jk} + \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \log p(\mathbf{x}_n|\mathbf{z}_n).$$

transition model

observation model

Expected Complete Data Log-likelihood

- The complete data log-likelihood takes the form:

$$Q(\theta, \theta^{old}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta^{old}) \log p(\mathbf{X}, \mathbf{Z}|\theta).$$

$$= \sum_{k=1}^{K} \gamma(z_{1k}) \log \pi_k + \sum_{n=2}^{N} \sum_{k=1}^{K} \sum_{j=1}^{K} \xi(z_{n-1,j} z_{nk}) \log A_{jk} + \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} \log p(\mathbf{x}_n|\mathbf{z}_n).$$

- Recall that in the E-step, we evaluate:

$$\gamma(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{X}).$$

$$\xi(\mathbf{z}_{n-1}, \mathbf{z}_n) = p(\mathbf{z}_{n-1}, \mathbf{z}_n|\mathbf{X}).$$

- In the M-step, we optimize *Q* with respect to parameters $\theta = \{\boldsymbol{\pi}, A, \boldsymbol{\phi}\}$.

Parameter Estimation: $\boldsymbol{\pi}$ and A

- Initial state distribution: Using Lagrange multipliers, the expected number of times in state *k* at time 1 is:

$$\pi_k^{new} = \frac{\gamma(z_{1k})}{\sum_{j=1}^{K} \gamma(z_{1j})}.$$

- Expected number of transitions from state *j* to *k* which begin at time *n*-1:

$$\xi(\mathbf{z}_{n-1,j}, \mathbf{z}_{n,k}) = p(\mathbf{z}_{n-1,j}, \mathbf{z}_{n,k} | \mathbf{X}),$$

  and the estimated transition probabilities work out to be:

$$A_{jk}^{new} = \frac{\sum_{n=2}^{N} \xi(z_{n-1,j}, z_{nk})}{\sum_{l=1}^{K} \sum_{n=2}^{N} \xi(z_{n-1,j}, z_{nl})}.$$

- The EM algorithm must be initialized by choosing starting values for $\boldsymbol{\pi}$ and **A**.

- Note that any elements of $\boldsymbol{\pi}$ or **A** that initially are set to zero will remain zero in subsequent EM updates.

Parameter Estimation: Emission Model

- For the case of discrete multinomial observed variables, the observation model takes the form:

$$p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\phi}) = \prod_{i=1}^{D}\prod_{k=1}^{K} \mu_{ik}^{x_{ni}z_{nk}}.$$

Same as fitting Bernoulli mixture model.

- And the corresponding M-step update: $\mu_{ik}^{new} = \dfrac{\sum_{n=1}^{N}\gamma(z_{nk})x_{ni}}{\sum_{n=1}^{N}\gamma(z_{nk})}.$

- For the case of the Gaussian emission model:

$$p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\phi}) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}}.$$

Remember:

$$\gamma(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{X}).$$

- And the corresponding M-step updates:

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k}\sum_n \gamma(z_{nk})\mathbf{x}_n, \quad N_k = \sum_n \gamma(z_{nk}),$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k}\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T,$$

Same as fitting a Gaussian mixture model.

Outline

- HMM learning and inference:
  1) Probability of an observed data sequence
  2) Learning the model parameters
  3) **Inferring the most likely state sequence**

Three problems and three solutions:

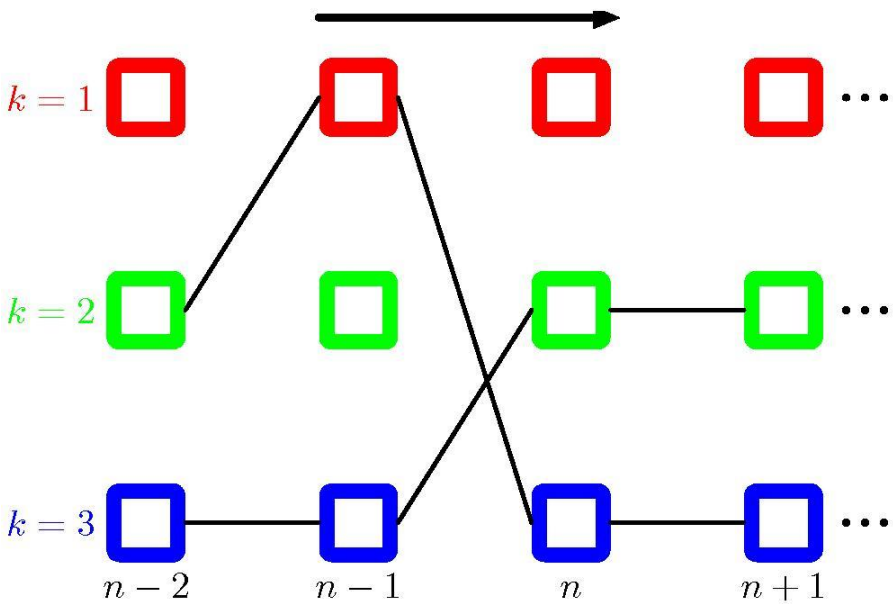
1. Computing probabilities of observed sequences: *Forward-backward algorithm*
2. Learning of parameters: *Baum-Welch algorithm*
3. Inference of hidden state sequences: *Viterbi algorithm* ←

The third problem: Viterbi Decoding

- The numbers $\gamma(\mathbf{z}_n)$ above gave the probability distribution over all states at any time.
- By choosing the state $\gamma^*(\mathbf{z}_n)$ with the largest probability at each time, we can make an "average" state path. This is the path with the maximum expected number of correct states.
- To find the single best path, we do Viterbi decoding which is a dynamic programming algorithm applied to this problem.
- The recursions look the same, except with 'max' instead of $\sum$.
- Same dynamic programming trick: instead of summing, we keep the term with the highest value at each node.
- There is also a modified EM (Baum-Welch) training based on the Viterbi decoding. Like K-means instead of mixtures of Gaussians.

Viterbi Decoding

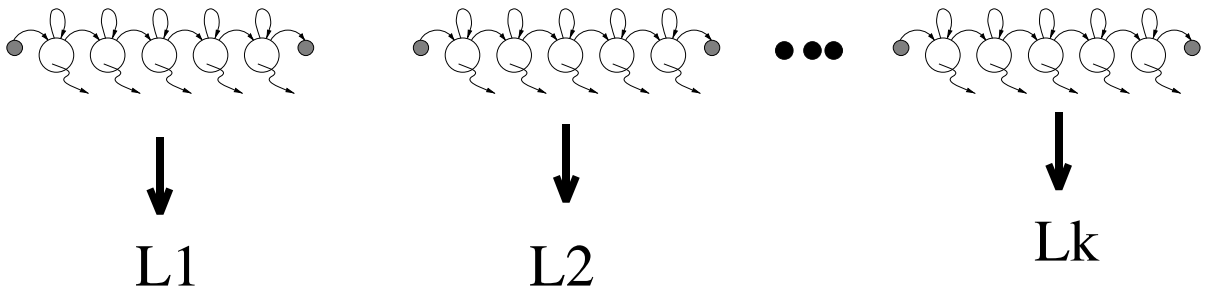
- A fragment of the HMM lattice showing two possible paths:



- Viterbi decoding efficiently determines the most probable path from the exponentially many possibilities.
- The probability of each path is given by the product of the elements of the transition matrix A_{jk}, along with the emission probabilities associated with each node in the path.

Using HMMs for Recognition

- We can use HMMs for recognition by:
  - Training one HMM for each class (requires labelled training data)
  - Evaluating the probability of an unknown sequence under each HMM
  - Classifying the unknown sequence by choosing an HMM with highest likelihood

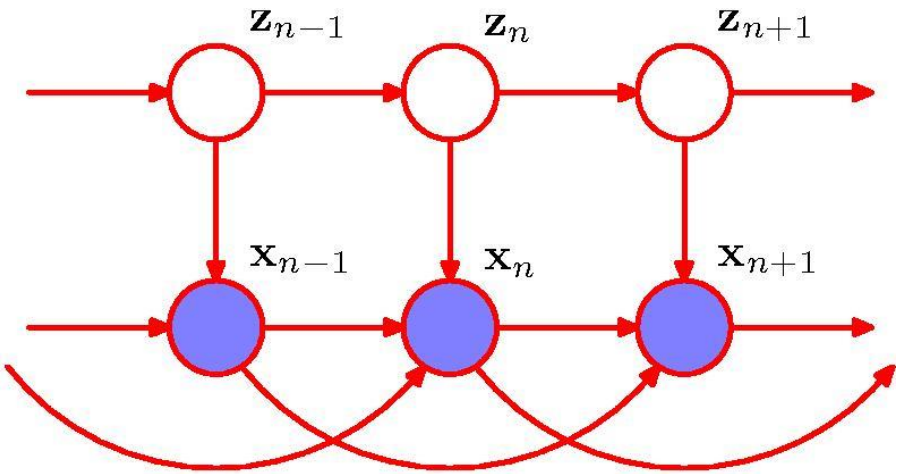


- This requires the solution of two problems:
  - Given a model, evaluate the probability of a sequence.
    (We can do this exactly and efficiently.)
  - Given some training sequences, estimate the model parameters.
    (We can find the local maximum using EM.)

Bonus

Autoregressive HMMs

- One limitation of the standard HMM is that it is poor at capturing long-range correlations between observations, as these have to be mediated via the first order Markov chain of hidden states.



- Autoregressive HMM: The distribution over $\mathbf{x}_n$ depends also on a subset of previous observations.
- The number of additional links must be limited to avoid an excessive number of free parameters.
- The graphical model framework motivates a number of different models based on HMMs.

Outline

- **HMM learning and inference:**
  1) Probability of an observed data sequence
  2) Learning the model parameters
  3) Inferring the most likely state sequence

- **Thursday:** Message passing
  - Introduction
  - Sum-product algorithm